Deborah J.
Mayhew &
Associates

Investing in Requirements Analysis

By Deborah J. Mayhew, PhD

While an in depth usability requirements analysis effort might be overkill in a development project involving a simple, content only website, intranets, web-enabled applications and websites providing rich transactional functionality are more like traditional desktop software and will benefit from usability techniques developed in the days before the World Wide Web.

In my experience as a usability engineering consultant, requirements analysis is still often a hard sell. While clients understand design, and seem to intuitively grasp the value of objective usability testing, they are often impatient to get to design, and reluctant to invest time and money in requirements analysis. Or, they believe they already understand requirements, through traditional systems analysis techniques. My goal in this article is to demonstrate the importance and payoff of investing in usability requirements analysis tasks by providing real world examples of the results of investing - or not investing - in this part of the Usability Engineering Lifecycle.

The Usability Engineering Lifecycle (Mayhew, 1999) documents a structured and systematic approach to addressing usability within the product development process. It consists of a set of usability engineering tasks applied in a particular order at specified points in an overall product development lifecycle. Several types of tasks are included in The Usability Engineering Lifecycle, as follows:

- Structured usability requirements analysis tasks
- An explicit usability goal setting task, *driven directly from requirements analysis data*
- Tasks supporting a structured, tops-down approach to user interface design that is *driven directly from usability goals and other requirements data*
- Objective usability evaluation tasks for iterating design towards usability goals

There are three main things that must be studied and understood in order to tailor user interface design to support unique usability requirements:

- The users
- The users' tasks
- The users' work environment

Traditional systems analysis usually (but not always) results in the inclusion of all required data and low-level functions, and structures them in a robust *implementation*

88 Panhandle Road
Post Office Box 248
West Tisbury, MA 02575
PH: 508-693-7149
FX: 508-693-9726
Email: drdeb@vineyard.net
Website: http://djmassoc.com

architecture. It usually fails to result in an *organization and presentation* of that data and functionality in a manner that *supports and optimizes the work performance of real users in their real work environment*. This missing piece is the whole point of a usability requirements analysis.

Thus, the purpose of usability requirements analysis tasks is to supplement more traditional types of systems analyses in order to define usability requirements in particular, and to point towards ways of meeting those requirements. Then, in later Lifecycle tasks, these requirements can be applied directly to making appropriate user interface design decisions.

Here are some examples demonstrating the value of investing in usability requirements analysis.

User Requirements

For two of my clients, I first interviewed project team members (developers and user management) to get a general sense of the internal target user population in order to design a questionnaire that I would later use to solicit user profile data directly from the users themselves. In one case, the project team was convinced that their internal users would have a generally low level of familiarity with the Microsoft Windows platform they planned for their product, and they were thus prepared to depart significantly from the Windows platform user interface standards in their product user interface design. The User Profile questionnaire, however, revealed a generally high level of Windows experience. This - and the fact that the Windows user interface standard was a good fit for the application functionality - led me to strongly advise them to adopt the Windows standards as closely as possible. They were still interested in creating their own unique user interface, but early testing of several alternative designs that varied in how faithfully they followed Windows standards clearly showed that users learned much more quickly, the more consistent the design was with Windows standards.

On the other project, the development team felt quite confident that their internal users would generally have high levels of computer literacy. An extensive User Profile questionnaire of over 800 users however, revealed that only a very small percentage of potential users had any experience with computer software at all, let alone the Windows user interface standards. In this case, based on this User Profile data, we designed (and validated through testing) a very simplified user interface that departed significantly from the Windows standards.

In both these cases, two things are clear: that project team members often have serious misconceptions about the important characteristics of their users, and that these misconceptions could have led them to design non-optimal user interfaces for those users.

Work Environment Requirements

An example of the importance of understanding the users' work environment comes from a project I worked on with a large metropolitan police department. Requirements analysis research revealed that in the typical police station, the appearance of the interior is usually dark, run down and cluttered, the lighting isharsh and artificial, and the air is close and sometimes very hot. The noise level can be high, the work areas are cramped and cluttered, and the overall atmosphere is tense and high-pressure at best, chaotic and sometimes riotous at worst. These conditions most likely have a general impact on morale, and certainly will have an impact on cognitive functioning that in turn impacts productivity and effectiveness. A user interface must be carefully designed to support the natural and possibly extreme degradations of human performance under these conditions.

In addition, it was observed that in the noisy, stressful and distracting work environment in a typical police station, users would frequently be interrupted while performing tasks, sometimes by other competing tasks, sometimes by unexpected events, unpredictable prisoners, etc. A user interface in such an environment must constantly maintain enough context information on the screen so that when users' attention is temporarily drawn away from their tasks, they can quickly get re-oriented and continue their task without errors, and not have to back up or repeat any work. Had we not understood these aspects of the work environment, we would likely have designed an interface that simply did not support these unique demands on users' cognitive resources.

Task Requirements

Besides the users themselves, and the environments they work in, the tasks they do also have their own inherent requirements.

One compelling example of the need for a thorough understanding of users tasks in order to achieve usable design comes from one of the earliest books on computer human interaction (Rubinstein and Hersh, 1984.) "For example, recently a water district in Maine installed an on-line billing system so that when a customer called in, the clerk could quickly retrieve a record of usage and billing for that customer. The managers noticed that the clerks were less than happy with the new system. A little investigation revealed that with the manual system, the clerks would write pertinent information in the margins of the records, such as that one person pays on the fifteenth of the month rather than on the first, or that the meter reader should contact the neighbor across the street for access to a house. This informal information was critical to the smooth operation of the office, but because there was no provision for it in an official field of the payment record form, the information was lost in the conversion to the on-line system. Understanding that there is informal as well as official information and recognizing the importance of the former would have reduced the disruption in work style."

One last example from my recent experience illustrates the importance of studying users' work in depth in order to design the user interface to a software application that supports that work. In this example, the application was a database system intended to support users in a government agency. The mission of the agency was to uncover and prosecute criminal behavior of a specific type. The job of the users of the intended system was to manage publicity for their agency in the media. These users needed to interact with the media to enhance reporting on criminal cases for the purpose of using the media to help discourage criminal behavior and encourage public cooperation in reporting and investigating criminal behavior.

The database of information being provided to these users included information on criminal cases in progress relevant to this agency, prior press releases and news articles on relevant criminal cases, a "phonebook" of contact information for reporters and other contacts in the media, records of past communications with the media, etc. A card sorting exercise revealed that users considered these categories of data types - Cases, Documents, Media Contact Info, Media Communications - to be distinct and meaningful categories that would provide a good foundation for the navigational structure of the application. However, what was not revealed by the card sorting exercise but *did* become clear through other methods of studying users' real tasks, was that when users searched for and found a particular item in one of these categories - such as a Case - they then typically wanted to automatically see items in other categories related to the item they had initially looked up.

Based on the card sort data, I had initially designed a navigational structure in which the user first selected a category of data, then searched for items in that category. Assuming from the card sort data that a search for one type of data was independent from searching for another type, I designed the interaction such that searching for something within any category of data was independent of searching for something within any other category of data. That is, if the user searched for and found a Case, and then navigated to the category of Documents, or to the category of Media Contact Information, the initial design assumed that they wanted to start an independent, unrelated new search in that category. However, further discussion with users revealed that in fact when the user searched for and found a Case, it was most likely that they would then want immediate access to Documents *related to that Case*, or Contact Information *related to that Case*. Similarly, if the user initially searched for a particular reporter's contact information, then they most likely would be interested in then seeing all Documents *related to this reporter*, and/or all Cases *related to this reporter*. This required a very different navigational and interaction design. Even though card-sorting data had been collected, without the ongoing input of users during early design, it would have been very easy to design the application is such as way as to make the users' most typical type of task very tedious and cumbersome.

Clearly, in cases like these, if an in-depth requirements analysis is not conducted prior to design and incorporated into design, it is possible - even likely - to design a user interface that does not optimally support users, their tasks or their work environment. Even if you have all the right functionality and data, an interface that does not directly support the unique requirements of the application is not usable. Finding out prior to design what the unique requirements are, and designing to support them, is much more cost-effective in the long run than finding out after launch that your design does not meet requirements.

Bibliography

Mayhew, Deborah J., *The Usability Engineering Lifecycle* , (Morgan Kaufmann Publishers, 1999)

Rubinstein, Richard and Hersh, Harry, *The Human Factor: Designing Computer Systems for People* , Digital Press, 1984, p. 26